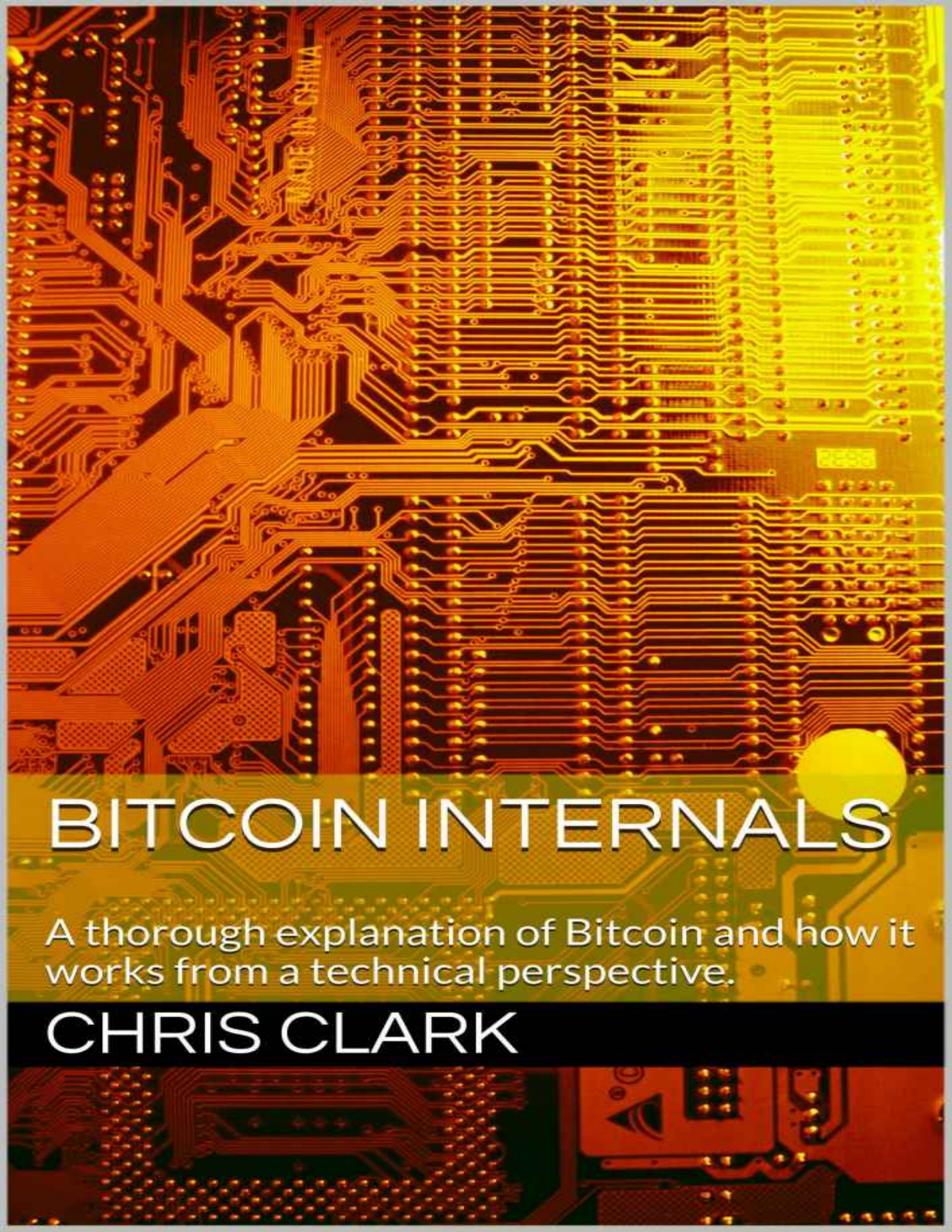
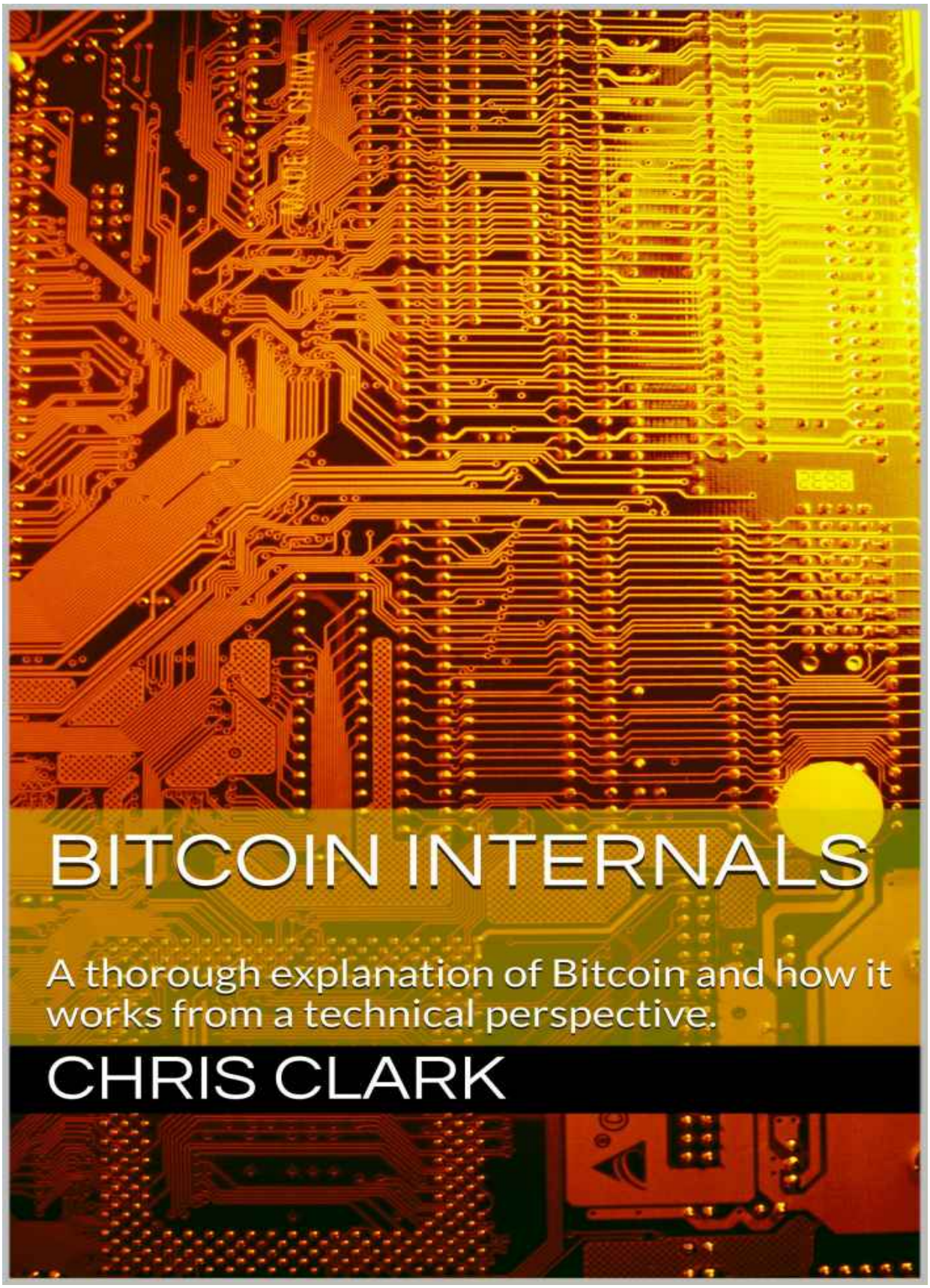

BITCOIN INTERNALS
A thorough explanation of Bitcoin and how it works from a technical perspective.
CHRIS CLARK


BITCOIN INTERNALS
A thorough explanation of Bitcoin and how it works from a technical perspective.
CHRIS CLARK

Bitcoin Internals

Chris Clark

July 31, 2013

Contents

Acknowledgements

I would like to thank Lucy Fang, Vadim Graboys, Dan Gruttadaro, VikingCoder, and Sheldon Thomas for their assistance in the preparation of this book.

Chapter 1
Introduction

Bitcoin is the world's first decentralized digital currency. Unlike most existing payment systems, it does not rely on trusted authorities such as governments and banks to mediate transactions or issue currency. With Bitcoin,

- Transaction costs can be reduced to pennies (in contrast to typical credit card fees of 2%).
- Electronic payments can be confirmed in about an hour without expensive wire transfer fees, even internationally.
- There is a low risk of monetary inflation[1] since the production rate of bitcoins is algorithmically limited and there can never be more than 21 million bitcoins produced.
- Payments are irreversible (there are no chargebacks), so there is a reduced risk of payment fraud.
- Payments can be made without identification, though some extra effort is needed to ensure that one's identity cannot be exposed (See Section 2.1).
- Responsibility is shifted to the consumers, who can permanently lose all of their bitcoins if they lose their encryption keys.

What is a bitcoin? A bitcoin is basically a digital record in a public ledger that keeps track of ownership in the Bitcoin system.[2] The ledger records ownership without revealing any real identities by using digital *addresses*, which are like pseudonyms. Ownership depends on possession of a secret digital key that gives the owner the exclusive ability to transfer bitcoins to other addresses. The owner can spend bitcoins to purchase goods and services from any business that chooses to accept them.

Who operates Bitcoin? There is no company or organization that runs Bitcoin. It is run by a network of computers that anyone can join by installing the free open-source Bitcoin software. The system is designed such that malicious attackers can participate but will be effectively ignored as long as the majority of the network is still honest. If attackers ever acquired the majority of the computing power in the network, they could reverse their own transactions and block new transactions while they held the majority, but they still wouldn't be able to steal bitcoins directly. People have an incentive to join the Bitcoin

network because those who process transactions are rewarded with newly created bitcoins.

Who created Bitcoin? Bitcoin started as a free, open-source computer program written by an author or group of authors who used the pseudonym Satoshi Nakamoto. The pseudonym was used in both the source code[3] and in the white paper that describes the idea.[1] Nakamoto's possible motivations for creating Bitcoin can be gleaned from some of his or her discussions on mailing lists:

> "[Bitcoin is] very attractive to the libertarian viewpoint if we can explain it properly. I'm better with code than with words though." - Satoshi Nakamoto[8]

It is estimated that Nakamoto now owns over $100 million worth of bitcoins, as of May 2013.[9] Nakamoto's involvement with the Bitcoin project faded in mid-2010, after which the open-source community, headed by Gavin Andresen, took over responsibility for the source code.[2]

Why do bitcoins have value? People consider bitcoins to be valuable for a variety of reasons.

- Utility: Bitcoins can be used to buy goods and services, most notably drugs on the Silk Road, where other currencies are not accepted.
- Exchange Value: Bitcoins can be traded for other currencies on exchanges such as Mt.Gox.
- Speculation: Bitcoin's popularity has been surging, and its value has surged along with it. Speculators pay for bitcoins in the hopes of making quick profits.
- Scarcity: The supply of bitcoins is limited. Production is algorithmically limited and is capped at 21 million bitcoins.

Historically, most currencies have been backed by either commodities or legal tender laws. Bitcoin is backed by absolutely nothing, so one might question whether its value is sustainable. There is one case of a currency that continued to function after its legal tender status was revoked: the Iraqi Swiss dinar.[5] After the Gulf War, the Iraqi government replaced Swiss dinars with Saddam dinars, but the Swiss dinars continued to circulate in the Kurdish

regions of Iraq due to concerns about inflation of the new notes. This example demonstrates that it's possible for a currency like Bitcoin to maintain its value.

Will Bitcoin succeed? There are two primary threats to Bitcoin's success: government intervention and competition. Of the two, competition is probably the bigger concern, as discussed below.

Bitcoin is famous for being a facilitator of illegal activities such as drug dealing and gambling.[4] The pseudonymous nature of Bitcoin makes it more difficult to use money-tracking methods to catch bitcoin-based drug dealers, gamblers, money launderers, and criminals. In the long run, if Bitcoin begins to replace the dollar, the feasibility of enforcing an income tax may become a major concern since bitcoin income can easily be hidden. Governments may decide that these concerns constitute grounds for banning Bitcoin.

There was a case in 2009 where the US Government successfully prosecuted a company that was producing a gold- and silver-backed private currency called "Liberty Dollars". The case was based on the charge that the liberty dollars resembled and competed with US dollars. Bitcoin, however, could not be dealt with in the same way since bitcoins don't resemble US dollars at all. Plus, there would be nobody to prosecute.

It would be quite difficult to enforce a ban on Bitcoin due to its distributed nature. Even if a ban worked, it would just push Bitcoin underground in the country that banned it. The system would still continue to operate normally in countries without a ban, and underground users would find ways to avoid being caught (by using the Tor service, for example).

A more likely threat to Bitcoin's success is its competition. Since the introduction of Bitcoin, several alternative currencies have sprung up. These alternatives claim to have advantages over Bitcoin, though none yet rival Bitcoin in popularity. Bitcoin definitely has the first-mover advantage, but if a competitor manages to become noticeably superior, there could be an exodus from Bitcoin. Commentators have criticized Bitcoin in various ways, most notably on its inability to scale to larger transaction volumes. However, Bitcoin developers are actively improving the system and these criticisms could be addressed before competitors get off the ground.

How safe is it to hold bitcoins? The value of a bitcoin has been quite volatile. The first purchase made in bitcoins was for two pizzas at a price of 10,000 BTC (BTC is the currency code for bitcoins). At bitcoin's current price level, those pizzas would have cost about a million dollars. Since there is no fixed exchange rate, the value of bitcoins can fluctuate greatly based on people's perceptions of their value. The price, shown in the chart below[5], has gone from $0 all the way up to $266. After it reached this peak on April 10th 2013, it crashed to below $60 on April 12th. And this wasn't the only time the price crashed. There was also a 68% drop between June 8th and 12th, 2011, and a 51% drop between August 17th and 19th, 2012.[10]

Despite this extreme volatility, the price has trended upward and will likely continue in this direction if Bitcoin sees further adoption. So while holding bitcoins is by no means a safe investment, it has the potential to be a good investment.

Chapter 2
Using Bitcoin

2.1 Wallets

To get started with Bitcoin, you need a wallet to hold your bitcoins. See

```
bitcoin.org/en/choose-your-wallet
```

for a list of options. The options for obtaining a wallet are:

- Running a bitcoin client on your computer or smartphone (clients come with wallets).
- Using a service that manages your wallet for you.

Using a service may be somewhat easier, but you really have to trust the service because they can potentially lose or steal your bitcoins. Since transactions are pseudonymous, they could even steal your bitcoins and tell you they lost them and you wouldn’t know the difference! So it is recommended that you run a Bitcoin client. There are several clients available currently. The original Bitcoin client is called Bitcoin-Qt or the "Satoshi Client". The rest of this section will assume that you are using the Bitcoin-Qt client.

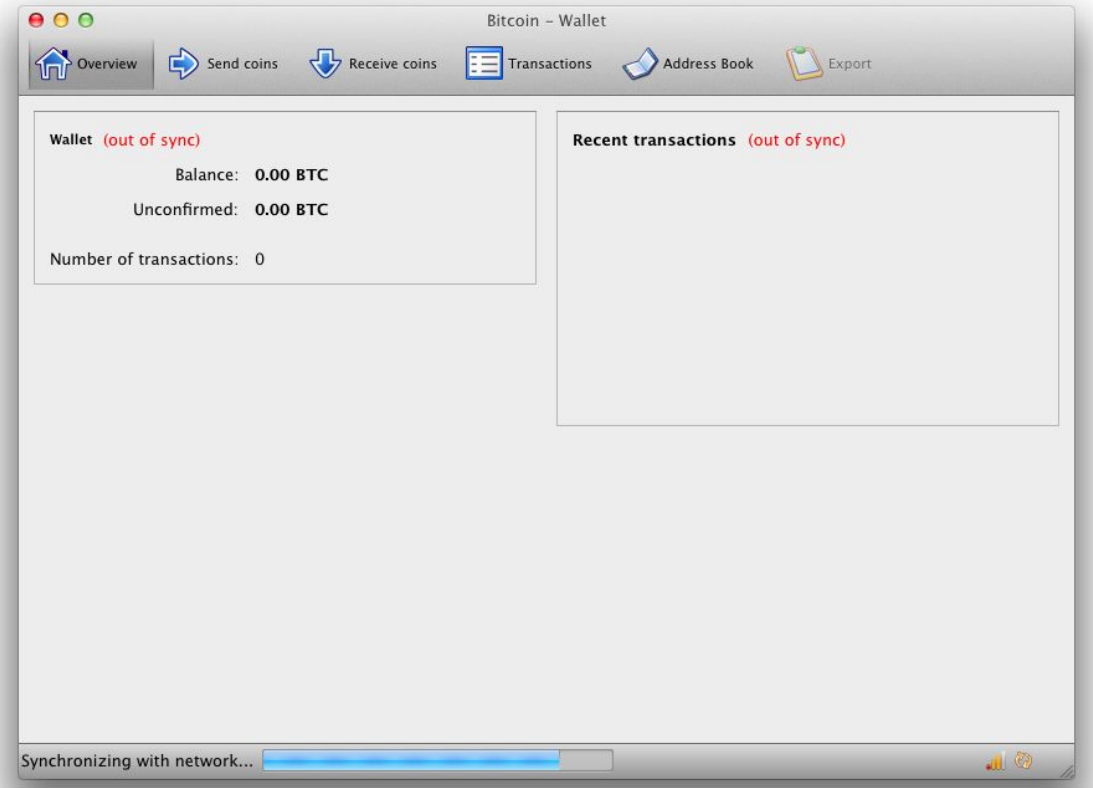


Figure 2.1: The overview tab of the Bitcoin-Qt client that shows the balance in your wallet.

The first time you run the Bitcoin-Qt client, it will create a wallet for you automatically. A wallet is a file that contains a set of addresses and keys that can be used to send or receive bitcoins.

An address is like a bank account number, except you can easily make as many as you want so there is no need to limit yourself to just one. Addresses are 27-34 character case-sensitive codes that look like this:

```
1tRkMBDDMGDLLcyh13eqP3WtUdcPxg66X
```

You could choose to use just one address, but it is important to realize that all transaction data for Bitcoin is public, so somebody could find patterns in the transactions going to and from that address. It is possible that these patterns could be used to reveal your identity. If you use many addresses though, there

won't be any patterns to find. Note that even though the addresses are publicly visible, nobody knows who owns which addresses, so you can still effectively maintain your anonymity. Theoretically, the government or a skilled hacker could link your Bitcoin address to an IP address and get your identity from your internet service provider. If you are worried about that, you should use Bitcoin with the Tor service, which would make it nearly impossible to find your IP address.[6] Also, to be anonymous you would have to be very careful about how you buy and sell your bitcoins and only use untraceable payment methods like cash.

The keys in the wallet are cryptographic codes that enable transfer of bitcoins from your addresses to other addresses. The keys look like addresses, but they are longer, containing 51 characters. If someone else gets access to the keys in your wallet, they can steal your bitcoins by transferring it to one of their addresses. If a hacker can hack into your computer, then they can probably get your keys, so make sure your system is secure. Some people don't feel safe keeping their wallet on their PC for this reason. Fortunately, more secure ways of storing wallets are available.

Hardware wallets are offline electronic devices that store keys in a micro-controller's memory. Because they are not connected to the internet and often require a user's confirmation of each transaction on the device, they are much harder to hack. To use a hardware wallet, you setup a payment on your computer and then plug the device into a USB port. The software on your computer will request the keys from the device and wait for you to confirm the transaction by pressing a button on the device. Then the device sends the keys to the computer to execute the transaction.

The most secure type of wallet is probably the paper wallet, which is just a piece of paper with keys written on it. The main downside of a paper wallet is that it is less convenient because you have to type in a long string of characters every time you want to make a payment. If you choose to make a paper wallet, you should still be careful about how you make it. For example, printing it on a printer may be unsafe. Sometimes printers will store data in their internal memory, which can be hacked.

Theft is not the only risk factor with wallets; you also have to be very careful to not lose your wallet. If you lose your keys, you lose all your bitcoins, so good backups are very important. If you are using a new address for every

transaction, it can be difficult to backup every address individually. A good solution is to use a *deterministic wallet*, which allows you to generate unlimited addresses from a single *seed* code. If you use a deterministic wallet, you only have to backup one code for all of your transactions because all keys and addresses can be regenerated from the seed code. The Armory and Electrum Bitcoin clients both use deterministic wallets, though the Bitcoin-Qt client does not.

2.2 Funding Your Wallet

Now that you have a wallet, the next step is to fill it with bitcoins. The simplest option is to use a service that accepts currencies such as US dollars and sends bitcoins to an address that you provide. If you want to get the best deal, you should use an exchange. An exchange lets participants submit orders to buy or sell bitcoins at specified prices, or just execute an order at the current market price. The prices can be expressed in a variety of other currencies such as US dollars or Japanese Yen. Currently, the biggest exchange is Mt.Gox. It charges transaction fees of between 0.25% and 0.6% depending on your 30 day trading volume.[7]

There are many other ways of obtaining bitcoins. BitInstant is popular service that accepts cash for bitcoins through MoneyGram agents at retail locations (CVS, Walmart, Grocery stores, etc.) for about a 4% fee. Another service called Coinbase allows you to buy or sell bitcoins using direct transfers to/from your bank account for a 1% fee. You can even buy and sell bitcoins through Craigslist or LocalBitcoins by meeting in person and paying cash.

If you are a business owner and just want to accept bitcoins, you can fill your wallet by publishing a Bitcoin address and requesting that customers send funds to that address.

Mining, the means by which bitcoins are initially put into circulation, provides another way of obtaining bitcoins. When mining, you get paid bitcoins to run a computer that processes transactions for the bitcoin network. Mining will be discussed more in Chapter 9.

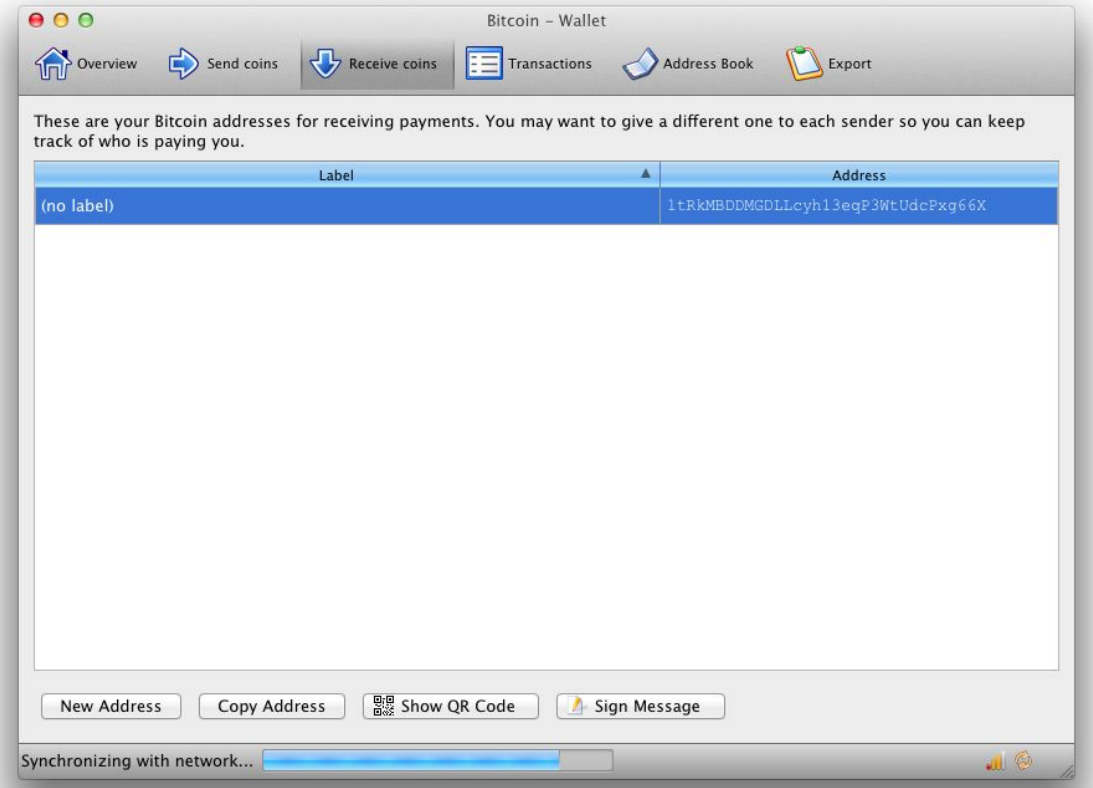


Figure 2.2: The "Receive coins" tab of the Bitcoin-Qt client where you can manage your addresses.

2.3 Sending Payments

Once you have bitcoins in your wallet, you will be able to see the balance in your wallet on the Overview tab of the Bitcoin client. You can then use the Bitcoin client to send funds to any other Bitcoin user. All you need is one of their addresses. Take the destination address and enter it in the "Send coins" tab along with the quantity you want to send. You don't have to worry about mistyping the address because it has a built-in checksum; if there is a typo in the address, the client will detect it and reject the payment.[8] The quantity field has 8 digits to the right of the decimal point so that bitcoins are divisible to a granularity of 1/100,000,000th of a bitcoin, a quantity known as a *Satoshi*. After you press the send button, the network will spend about an hour confirming the transaction. When the confirmation is complete the receiver

will see their confirmed balance go up. Bitcoin is not ideal for in-store transactions because of the long confirmation time, but merchants are still free to accept partially confirmed or unconfirmed transactions, which effectively trades fraud-resistance for speed.

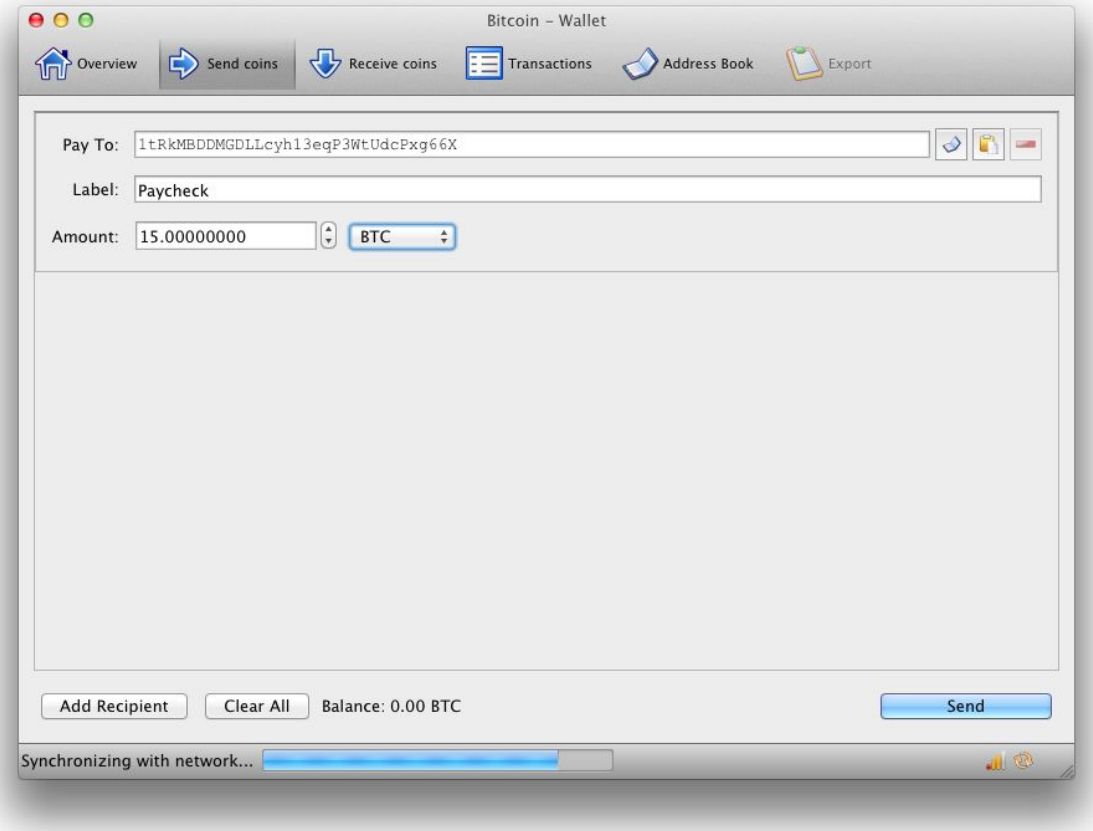


Figure 2.3: The "Send coins" tab of the Bitcoin-Qt client where you can make payments.

Chapter 3
Cryptography

A digital payment system like Bitcoin requires strong security against fraud. This chapter introduces the cryptographic technologies that Bitcoin utilizes to ensure that the system can't be foiled by hackers.

3.1 Cryptographic Hash Functions

When transmitting data over a network, it is very important to make sure that the data is not corrupted during transmission. A *cryptographic hash function* can help solve this problem. A cryptographic hash function takes a sequence of bytes and computes a fixed-length value based on the data, called a *hash* or *digest*, with some special properties:

- it is easy to compute the hash for any input
- changing any bit in the input produces a completely different output
- it is not feasible to find any input that corresponds to a given output or any two inputs with the same output

The hash function used in Bitcoin is called SHA-256. The output of SHA-256 is 256 bits, or 32 bytes. Here are some examples (0x at the beginning of a number means that the number is expressed in hexadecimal notation):

```
sha256('Bitcoin') =
0xb4056df6691f8dc72e56302ddad345d
65fead3ead9299609a826e2344eb63aa4

sha256('bitcoin') =
0x6b88c087247aa2f07ee1c5956b8e1a9
f4c7f892a70e324f1bb3d161e05ca107b
```

Even though the only difference in the input is a change in the case of the first letter, the outputs are unrecognizably different.

Hash functions are not just useful for guarding against transmission failures; they are also valuable for preventing tampering of input data. For example, let's say Alice and Bob are moving into a two-bedroom apartment together, but one bedroom is bigger than the other. They both prefer the bigger

room, but it isn't clear how much more one should have to pay for it. Neither one wants to be the first to suggest a number because it would weaken their bargaining position. So they agree to the following protocol:

1. Write down a bid for how much they would pay per month to live in the bigger room.
2. Place the bids face down on the table without showing the other person.
3. When both bids are on the table, flip the papers over to reveal the bids.
4. The higher bidder gets the bigger room. The price the winner pays is the average of the two bids.

With this protocol, both parties are guaranteed to get a deal that is better than or equal to what they bid for.

Now let's say that Bob is on a trip, so this process has to be done remotely. If they try to negotiate over the phone or email, there is no guarantee that both sides will announce a number at the same time. Hash functions can solve this problem. Alice and Bob can each write down a sentence like "I'll pay $650" or "$700 is my bid", take the hash, and email the hash to the other person. At this point, neither knows the bid of the other, but the bids are locked in. If one of them tries to change their bid, they would have to find another sentence that matches the given hash, which is not feasible. After exchanging hashes, Alice and Bob exchange the sentences containing their bids and compare. They each rehash the sentence of the other to verify that the bid wasn't changed since the hash exchange. If one of them finds that the hash doesn't match, they will know it's time to start looking for a more honest roommate.

Given a hash of a piece of data, it is possible to later confirm that the data was not tampered with by rehashing the data and making sure that the hash comes out the same.

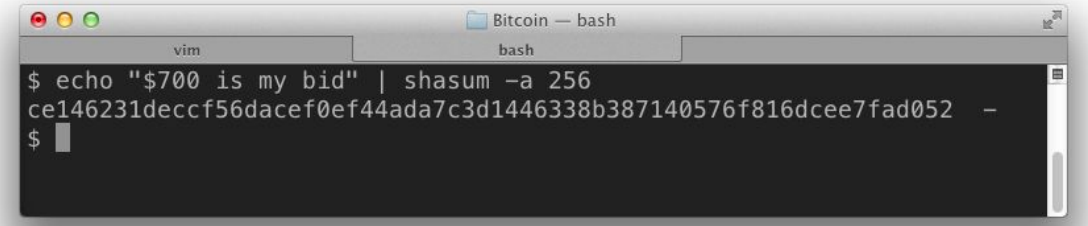


Figure 3.1: Calculating the SHA-256 hash of a sentence using shasum in the Mac OSX Terminal. In the Linux terminal, sha256sum can be used.

3.2 Merkle Trees

Cryptographic hash functions are often used to verify the integrity of a list of items, such as the broken-up chunks of a large download. In such cases, one option is to merge all the chunks and take the hash of the complete download. The problem with this is that if one chunk is corrupted, the user won't find out until the entire download is complete, and even then they won't know which chunk is corrupt. A better solution is to take the hash of each chunk individually so that each chunk can be verified as it comes in. However, if there are a large number of chunks, then there is a greater chance that some of the hash values will become corrupted. Furthermore, this is a lot of data for the trusted source to store. Ideally, a trusted source would only have to provide one hash, and the rest of the hashes could be downloaded from untrusted sources, such as peers in a peer-to-peer network. This can be accomplished using a *top hash* generated by hashing all of the hashes of the chunks. The resulting structure is called a *hash list*.

If the number of chunks is very large, the list of hashes of all the chunks might also be quite large. In order to verify just one chunk against the trusted top hash, one would need to obtain all of the hashes in the hash list. Ralph Merkle proposed the idea of a *hash tree* in 1979, which allows a chunk to be verified with only a logarithmic number of hashes.[6] In a hash tree, or *Merkle tree*, hashes are taken of all chunks as in a hash list, but then these hashes are paired and the hash of each pair is taken, and these hashes are then paired again, and so on until there is only one hash at the top of this tree of hashes.

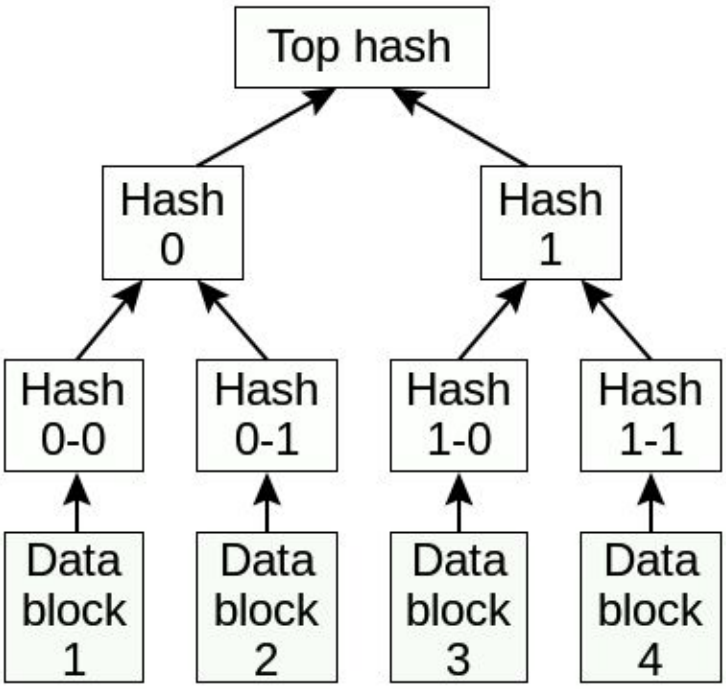


Figure 3.2: The structure of a Merkle Tree.

To verify the integrity of just one chunk, it is only necessary to obtain a small subset of the hashes in the hash tree. For any hash in the tree, if the desired chunk is not in the branch below it, then that branch can be stubbed out by dropping it and keeping only the hash at the top of that branch. For example, if you wanted to verify data block 1 in the diagram, you need Hash 0-0, Hash 0-1, Hash 0, Hash 1 and the Top hash. The branch rooted by Hash 1 can be stubbed out, removing Hash 1-0 and Hash 1-1, keeping just Hash 1 to represent the whole branch. For large trees, the number of hashes needed to verify one chunk can be much smaller than the number of chunks.

3.3 Public Key Cryptography

Bitcoin does not do any encryption; all transaction information is publicly visible. However, it does rely heavily on digital signing, which is a technology based on public key cryptography.

Earlier forms of cryptography were based on the idea of secretly sharing an encryption/decryption key that would be kept private at all times. This method, known as *private key cryptography*, is useful in situations where two parties can communicate privately at one point in time and want to be able to securely communicate over an insecure channel, like radio or the internet, at a later point in time. The simplest and most secure encryption scheme is called *one-time pad encryption*. A *one-time pad* is a long string of bits (zeroes and ones) that serves as an encryption/decryption key. To encrypt a message, the one-time pad is lined up next to the bits of the message, and for any position where the pad has a 1, the corresponding bit in the message is flipped. To decrypt the encrypted message, the exact same procedure is used. It is called "one-time" because once a portion of the bit sequence is used, it is thrown out and never used again. One-time pad encryption is very simple and has been mathematically proven to be absolutely impossible to crack.[7] The only problem is that the two parties have to exchange the one-time pad without exposing it to anyone else. This may be fine if both parties can meet in-person, but it isn't very helpful for communicating over the internet.

Public key cryptography allows encrypted communication without private key exchange. If Alice and Bob want to talk securely, they can do so by agreeing to use the following protocol. First, they each run a special algorithm to produce a key pair consisting of a public key and a private key. They each keep their private keys secret but publish their public keys, which become visible to the whole world. The public and private keys have a special mathematical relationship that allows Alice to encrypt a message M using Bob's public key K_{pub} that only Bob's private key K_{priv} can decrypt. Letting C denote the encrypted message,

$$C = \text{encrypt}(M, K_{pub})$$

$$M = \text{decrypt}(C, K_{priv})$$

This is accomplished by using mathematical functions that are computationally intractable to invert. For example, it is very easy to multiply two large prime numbers, but it is much more difficult to find the prime factors of a product. The mathematical details are beyond the scope of this book, but search for "RSA" for more information.

Using public key cryptography, secure messages can be sent between individuals who have only ever had contact through insecure channels, such as the internet.

3.4 Digital Signatures

Key pairs in public key cryptography are complementary in that they can effectively be swapped. For normal encryption/decryption, the public key is used to encrypt and the private key is used to decrypt. If instead the private key is used to encrypt, then only the corresponding public key can be used to decrypt the result. This can be used to create a digital signature that proves that a particular individual sent a message. A recipient of a signed message can confirm that a message was not sent by an impostor (authenticity), was not tampered with (integrity), and can disprove any sender who denies sending the message (nonrepudiation). This is exactly what the Bitcoin systems needs to prevent fraudulent transactions.

To send a signed message with contents *M*:

1. Take the hash of *M*:

$$H = \text{sha256}(M)$$

2. Encrypt *H* with the private key to get the signature:

$$S = \text{encrypt}(H, K_{priv})$$

3. Send the signature *S* along with the message *M*

To verify that a signature *S* is valid for message *M*:

1. Take the hash of *M*:

$$H = \text{sha256}(M)$$

2. Decrypt *S* with the public key:

$$H' = \text{decrypt}(S, K_{pub})$$

3. Compare to see if $H = H'$. The signature is valid if they are the same.

Bitcoin uses a digital signature scheme called the Elliptic Curve Digital Signature Algorithm (ECDSA). The mathematics underlying the algorithm are rather complex. It is more complex than the more common RSA public key crypto-system, but it is considered to be more secure for a given key length.

Chapter 4
Digital Currencies

4.1 Properties

A secure digital payment system should have the following properties to prevent fraud:

1. Authenticity - Only the owner of a quantity of money can spend it
2. Security - Money can not be counterfeited (token forgery), and the owner can only spend it once (the "double-spending" problem)
3. Nonrepudiation - A recipient cannot deny receiving money

Nonrepudiation is not as crucial as the other two, but if the system did not have this property, it would be impossible to arbitrate disputes in which a seller denied receiving payment and refused to provide the merchandise.

There are also three optional properties that make the system more powerful:[9]

1. Anonymous - payer identification is not disclosed to payee or third parties (this can be broken down into three components: payer anonymity, untraceability, and unlinkability)
2. Offline - payee can be confident that they will receive funds from a transaction without immediately contacting a third party such as a bank
3. Decentralized - there is no trusted authority (e.g. bank) needed to process transactions

Digital cash is defined to be any digital payment system that satisfies properties 1-4.[11] Bitcoin doesn't completely satisfy property 4, so it is not technically digital cash, but it is close because it is pseudonymous.

4.2 Double-Spending

All electronic payment systems rely heavily on cryptography. Digital signing can easily be used to guarantee most of properties 1-3, but it doesn't help prevent the problem of "double-spending".

Double-spending is a type of digital payment fraud in which a user tries to spend the same money twice. For example, imagine Chuck has a debit card with $100 in his account. Now let's say he opens two tabs in his web browser, one for Amazon and one for Ebay. In each of these tabs, he adds a $100 item to his cart, enters his debit card number, and presses submit at almost the same time. If his bank had really bad security, both sites would see that he had $100 in his account and approve the purchase, yielding $200 worth of goods. This is a double-spend. For online centralized systems such as credit cards, detecting double-spending is easy since all transactions are seen immediately. For offline or decentralized systems, however, it is more difficult.

Solving the double-spending problem is the main hurdle that digital payment systems need to overcome. The tricky part about double-spending is that each payment would be completely legitimate if the other didn't exist. The only way to detect double-spending is to be aware of all transactions and look for duplicates.

After detecting a double-spend, there are a couple of options. One option is to reveal the identity of double spenders so that the victims can sue. Obviously, this isn't ideal because it would require a lot of legal overhead and would still require some trusted authorities in the system, even if only the courts.

The best option is to only consider the first transaction of a double-spend to be valid. Rejecting both double-spend payments would be bad because recipients would never have confidence that their incoming payments were secure. The sender could later double-spend and they would lose their funds. So in this case it is necessary that the system be able to determine which of two double-spend payments came first. Bitcoin's solution to this problem will be discussed in Chapter 8.

4.3 Types of Digital Payment Systems

Type 1. Credit/Debit Cards (Properties 1-3)

Most of the world is still operating with the most primitive type of electronic payment system: credit cards and debit cards. These transactions are "identified" because the merchant can see the owner's name on the card and the credit card company can track their purchases. These transactions are also "online," which means that merchants must contact a bank or credit card

company for every transaction to verify that funds are available. And these transactions are "centralized" because the system doesn't work without the credit card company or bank. This also means that if the credit card company or bank decides to freeze a user's account, the user would lose access to their money.

Type 2. Digital Cash (Properties 1-4)

In 1982, David Chaum published a paper called "Blind signatures for untraceable payments,"[12] which contained the first description of a digital cash scheme. In Chaum's proposed system, banks issue cryptographically signed digital notes that can be used anonymously like cash. Individuals may request digital notes from the bank for a specified amount of money. The bank then creates a set of special digital notes that only the bank can produce with its secret cryptographic key. Each note issued is worth a fixed quantity, say $1, and whoever has access to the note can spend it, so it can be stolen just like cash. When the bank sends the notes to the customer, it simultaneously deducts the corresponding quantity from the customer's bank account. At this point, the bank knows who they issued the notes to, but the customer then modifies the notes in such a way that the bank is not be able to trace them. However, even after the modification, the bank can still verify that they were in fact notes issued by that bank.

This is the magic of blind signatures that Chaum introduces. He explains it with an paper-based analogy. Let's say Alice is a customer at Chaum Bank and wants some paper blind signature notes. She goes to the bank and approaches the desk with the old-fashioned deposit slips. But at Chaum bank, there are also stacks of blind note forms, envelopes, and carbon paper. The blind note form just has a long sequence of empty boxes where Alice fills in random numbers to form a unique serial number, and a line for a signature that she leaves blank. She puts this paper and a slip of carbon paper in an envelope, seals it, and brings it to the teller. The teller asks for Alice's ID, signs the envelope with a special signature that indicates it is worth $100, and deducts $100 from Alice's account. As Alice leaves the bank, she opens the envelope and extracts the blind signature form that now has the bank's signature on the signature line because of the carbon paper. She can now spend this note at her favorite store. The merchant then takes the note back to the bank. The teller verifies the signature and makes sure that the serial number has not already been used to ensure that Alice didn't photocopy the note. (In the cryptographic case, Alice

can create an exact duplicate of the note, but she can't modify it in any way, so we don't have to worry about her changing the serial number after the bank signs it. Also, since Alice chooses a serial number randomly, it is nearly impossible for it to be a duplicate on accident.) If everything checks out, the bank credits the merchant's account with $100. Since the bank didn't see the serial number when Alice got the note signed, there is no way to tell that the merchant's note came from Alice. At the end of this process, $100 got transferred from Alice's account to the merchant's account without the bank knowing that Alice did business with the merchant, and without the merchant needing to know who Alice is. Of course, the merchant has to get confirmation from the bank before giving Alice her merchandise, so this is still an "online" system, but electronically this can be done almost instantly.

Type 3. Offline Digital Cash (Properties 1-5)

Offline payment systems have a significant challenge that online systems don't have. They have to allow transactions to clear without contacting the trusted authority, such as the bank. At first this seems impossible, because if a customer uses the same piece of digital cash at two merchants consecutively, and both of those merchants are disconnected from the rest of the system, then there is no way to tell that the cash was double-spent. But if the merchants could identify the customer, they could later sue the customer for the fraudulent transaction when they find out from the bank that the digital cash was double-spent. The only problem with this solution is that if the merchant can identify the customer, the system is no longer anonymous. It turns out there is a way to reveal the customer's identity only after a double spend. This is the idea first presented by Chaum, Fiat, and Naor in their 1989 paper "Untraceable Electronic Cash".[13]

The idea is to attach a set of K pairs of numbers to every digital cash note. Any single pair contains enough information to reveal the owner's identity, but one number from each pair is not enough to determine anything. Every time a note is spent, the merchant issues a challenge that requires one number from each pair. The merchant randomly chooses which number from each pair the customer must present. If two merchants randomly get one number from each pair for the same note, it is very likely that they will have at least one pair where they chose differently, so together they have the complete pair. When they both submit their records to the bank at a later date, the bank can combine the two parts of the pair to reveal the thief's identity.

Type 4. Decentralized Digital Currency (Properties 1-3, 6)

Bitcoin is the first decentralized digital currency. Making a decentralized system is significantly more challenging than making a centralized system, so some sacrifices had to be made on the other properties listed at the start of the chapter. Bitcoin requires network access for payment verification, so it is not offline and does not satisfy property 5. It is also not anonymous because all transactions are publicly announced, so it does not satisfy property 4 either. However, it is pseudonymous, which means that it is possible for users to avoid revealing their identity if they are careful. The rest of this book will explain how Bitcoin works.

Chapter 5
Precursors

5.1 Triple Entry Accounting

The way Bitcoin records ownership is based on an idea originally presented in 2005 by Ian Grigg called "Triple Entry Accounting".[14] In this system, payment receipts are not just verification that the ownership ledger changed, the receipts themselves are the ledger. It is called "triple entry" because the sender, receiver, and a third party all have a copy of the receipt, and any single copy provides sufficient proof of the transaction. Grigg explains how digital signatures can be used to sign the receipts so that they cannot be forged or repudiated. If Alice wants to send money to Bob, Alice creates a message that assigns a specified quantity of her money to Bob and signs the message with her private key, much like a paper check. This signed message is the receipt of the transaction and anyone can use this receipt to confirm that Alice made the payment.

The third party in Grigg's triple entry accounting system was a trusted authority that would issue money and verify that the sender had enough funds to make the payment. Bitcoin does not have any trusted authorities, so it must perform these checks differently.

5.2 Publicly Announced Transactions

Some of the concepts Bitcoin uses to maintain a decentralized ledger are based on a digital currency proposal that Wei Dai presented in 1998 called "B-Money".[15] In Dai's proposed system, every participant maintains a log of how much money belongs to each pseudonym and money is transferred by publicly announcing a digitally signed message. Both of these concepts are used in Bitcoin. Furthermore, Bitcoin's method of creating new money based on an investment of computation time is similar to Dai's, where solving a difficult computational problem constitutes a *proof of work* that is rewarded with new money.

5.3 Proof of Work

Both Dai and Nakamoto cite Adam Back's 1997 Hashcash protocol[16], which is a proof of work scheme originally designed to fight email spam. When using Hashcash, the sender's email client solves a difficult computational problem every time it sends an email. The receiver considers this evidence that the sender is not a spammer because it would be expensive for spammers to solve the problem for every email sent. Conceptually, this is similar to charging a small fee for every email sent, but instead of money, the fee is just a bit of computer time.

The idea of using a proof of work to fight email spam was originally published by Dwork and Naor in 1993.[17] Hashcash differs in the proof of work problem it uses. In Hashcash, the computational problem is to find a value that has a hash starting with a specified number of zeroes. This is convenient because it is very fast to verify and easy to probabilistically estimate how much work it takes to solve the problem, because the only known method is to try random values until a solution is found. Bitcoin adopted this proof of work problem with slight modifications.

5.4 Proof of Work Chains

Bitcoin's ledger also uses the concept of a chain of proofs of work, in which each proof of work depends on the results of previous proofs of work, creating an indisputable chronological ordering. This idea was introduced by Nick Szabos in his digital currency proposal called Bit Gold, which was released between 1998 and 2005.[18] A proof of work chain provides additional security because each individual proof of work gets buried under the subsequent proofs of work. As the chain grows, it becomes more and more difficult to redo all the proofs of work.

Chapter 6
Technical Overview

6.1 Architecture

Bitcoin is run by a peer-to-peer network of computers called *nodes*. Nodes are responsible for processing transactions and maintaining all records of ownership. Anyone can download the free open-source Bitcoin software and become a node. All nodes are treated equally; no node is trusted. However, the system is based on the assumption that the majority of computing power will come from honest nodes (see Chapter 8). Ownership records are replicated on every full node.

6.2 Ownership

There are two ways to track ownership of a currency:

1. Possession of a token that independently represents value (e.g. paper cash, metal coins, and Chaum's digital cash)
2. Possession of a key that controls access to records in a ledger (e.g. checks, credit cards, Paypal, and Bitcoin)

Designing a decentralized currency is challenging because it is not obvious how either of these methods can be used without some trusted authority like a bank. In method 1, somebody has to issue the tokens and in method 2, somebody has to maintain the ledger. If anyone can issue tokens or edit the ledger, then the system is bound to fail. Bitcoin uses the ledger-based method, but it manages to work due to a novel method that allows a distributed network of untrusted peers to maintain a trustworthy ledger.

Bitcoin's ledger, known as the *block chain*, can be thought of as a record of receipts for all transactions that have ever occurred in the Bitcoin system. Unlike a typical bank's ledger, it doesn't contain any account balances. Rather than recording a quantity of bitcoins for each owner, it records an owner for each quantity of bitcoins transacted. The owner is just the recipient listed on the transaction receipt in the block chain (until the bitcoins in that transaction are spent). To spend bitcoins, the owner creates a new transaction that takes the

bitcoins from a prior transaction (one that was sent to the owner) and assigns them to someone else. The owner is the only one who has the ability to create such a transaction because it requires their digital signature.

The previous transaction from which bitcoins are taken is called an *input*. When a transaction is used as an input, we say that the transaction itself is *spent* because all of its value will be sent to the recipient(s) and the transaction can never be used as an input again. If an input's value is greater than the desired payment amount, that is not a problem; transactions allow multiple recipients, so the owner can send a portion of the input's value back to one of their own addresses as change.

6.3 Addresses

As previously mentioned, the block chain doesn't record owners by their true identities; it uses codes called addresses (introduced in section 2.1). It is difficult or sometimes impossible to connect an address to a person unless the person allows it, which provides a degree of anonymity. To create an address, the Bitcoin client first generates a public-private key pair. The address is then formed by a applying the following function to the public key (in pseudocode, ++ represents concatenation):[19]

```
version = (1 byte version number)
keyHash = RIPEMD-160(SHA-256(publicKey))
data = version ++ keyHash
dataHash = SHA-256(SHA-256(data))
checksum = (first 4 bytes of dataHash)
address = Base58Encode(data ++ checksum)
```

The version byte represents the version of the address, which has the value of 0 for normal addresses on the main Bitcoin network.[10] The RIPEMD-160 function is a hash function with a 20 byte output. Base58Encode converts a binary value to a string of numbers and letters. Because the first byte in the input to Base58Encode is the version byte, which is 0, all normal addresses start with the digit "1" because 1 is the Base58 representation of 0.

The whole address generation function is a hash function itself, so addresses are just hashes of public keys. Bitcoin could have used public keys directly without hashing, but there are three benefits that hashed addresses have over public keys:[23]

1. Addresses are shorter than public keys.
2. Addresses have built-in checksums so that a mistyped address can be detected.
3. Addresses are more secure against cryptographic attacks. Even if someone came up with a way to reverse the supposedly one-way function in the elliptic curve digital signature algorithm, they would still need the public key to use it. With just the address, the only option is a brute-force attack (trying every possible private key until one works).

Chapter 7
Transactions

7.1 Structure

When a Bitcoin user initiates a payment, the client creates a transaction message and submits it to the peer-to-peer network. A transaction contains the following information, with CTxIn and CTxOut shown below:[11]

```
class CTransaction
    int nVersion;
    std::vector<CTxIn> vin;
    std::vector<CTxOut> vout;
    unsigned int nLockTime;
```

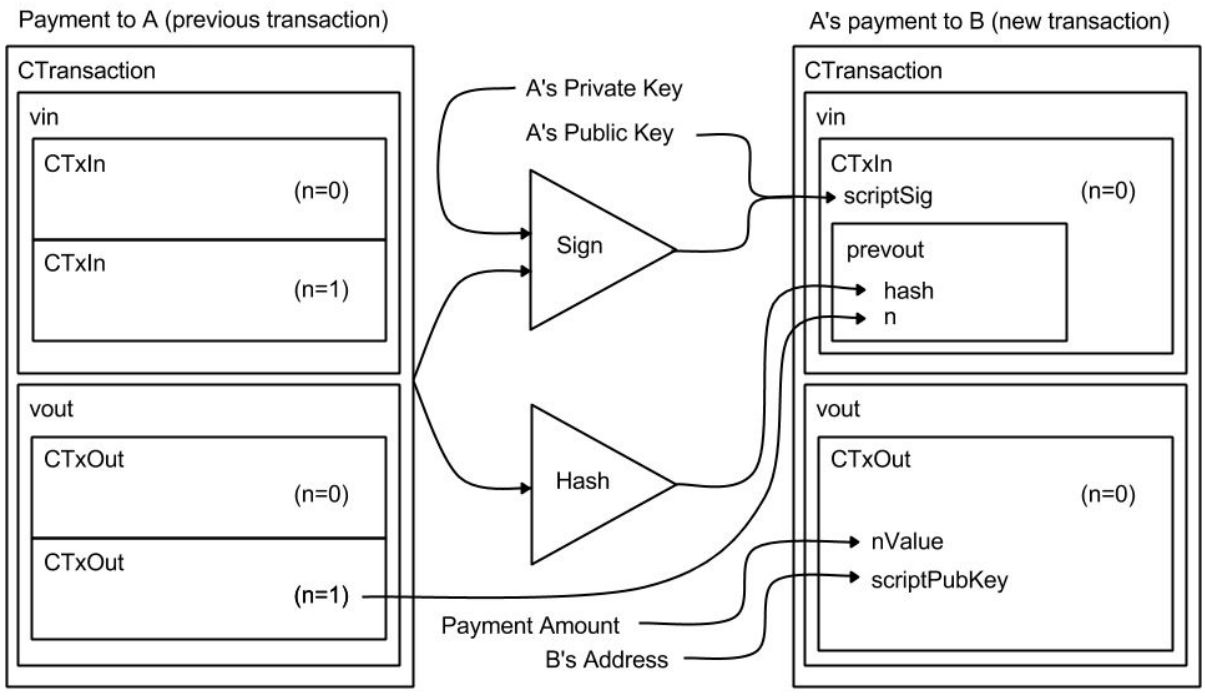


Figure 7.1: Diagram of a Bitcoin transfer where A is sending bitcoins to B. The transaction on the left shows multiple inputs (CTxIn) in vin and multiple outputs (CTxOut) in vout to illustrate that vin and vout are arrays. The labels "(n=0)" and "(n=1)" indicate the position in the array. The contents of the inputs (CTxIn) and outputs (CTxOut) on the left are hidden to simplify the diagram.

Basically, the transaction message lists a set of inputs (`vin`), which refers to previous transactions, and a set of outputs (`vout`), which specifies destination addresses. This allows a transaction to pull money from multiple sources, pool it together, and then distribute it to multiple destinations. Of course, any transaction that distributes more money than it pulls in will be considered invalid and will be ignored (except coinbase transactions that will be explained later). The `nVersion` variable is a version number for the transaction so that the peer-to-peer network can still function even when different nodes have different versions of the Bitcoin code running. The `nLockTime` variable can be used to prevent the transaction from being processed before a specified time, which is useful for making contracts.[12]

```
class CTxIn
    COutPoint prevout;
    CScript scriptSig;
    unsigned int nSequence;

class COutPoint
    uint256 hash;
    unsigned int n;
```

An input (`CTxIn`) references an output of a previous transaction (`prevout`), which is the output to be spent. In order to uniquely reference an output, it is sufficient to provide the hash of the transaction the output is in (`hash`) and the index of the output within that transaction's `vout` vector (`n`). This is precisely the data contained in `COutPoint`, the type of `prevout`. This output referenced in `prevout` will be completely spent. If the user wants to spend only a portion of the previous output, they can create an extra output in the new transaction that points back to their own wallet, just like receiving change in a cash transaction.

Inputs also specify a `scriptSig` that contains the owner's public key and a signature to prove that the spender owns the output being spent. The `CScript` type extends `std::vector<unsigned char>`, so it's a type of string class. The

`nSequence` variable, along with `nLockTime` is used for making contracts (see the footnote for `nLockTime`).

```
class CTxOut
    int64 nValue;
    CScript scriptPubKey;
```

A simple transaction output (`CTxOut`) specifies an address (in `scriptPubKey`) and how much to send to that address (`nValue`), specified in *Satoshis*, the smallest unit in the Bitcoin currency, worth 1/100,000,000th of a bitcoin. The script is customizable though, so it can be more complicated than just specifying a destination address. More complicated scripts can be used to create customized contracts that prevent a transaction from completing until certain conditions are met.

7.2 Verification

After a Bitcoin client submits a new transaction to the peer-to-peer network, other nodes in the network will start trying to process it into the block chain (this procedure will be explained in Chapter 9, "Mining"). The first step in this process is to make sure that the transaction is valid. The following checks are performed during validation:

- Ensure that transaction passes various sanity checks (bounds checking, syntactic correctness, etc.).
- Reject if this exact transaction is already in the block chain or pool of transactions waiting to be processed. This prevents the same transaction from being processed twice.
- For each input, concatenate the scriptSig script of the input with the scriptPubKey script of the output that it references, execute this script, and verify that the result is True. If these scripts are of the standard format, this will confirm that the owner of the bitcoin initiated the transaction.
- Reject if any of the transaction outputs specified in the transaction's inputs have already been used in another transaction in the block chain. This is used to prevent double-spending.
- Ensure that the sum of the input values is greater than or equal to the sum of the output values (if the inputs are greater, the difference is the transaction fee, which is paid to the miner of the transaction (see Section 9.4)).

The script execution step requires further explanation. The purpose of the script is to verify that the transaction was digitally signed by the owner (the person the previous transaction was sent to). Transactions only specify an address as a destination, but addresses are just hashes of public keys. So basically, the script's job is to make sure the hash of the provided public key is the destination address of the previous transaction and this same public key belongs to the user who signed the transaction. In python, the script would look something like this (bracketed names represent hard-coded values in the script):

```
def scriptSig():
  return <sig>, <pubKey>

def scriptPubKey(txHash):
  sig, pubKey = scriptSig()
  return (hash(pubKey) == <pubKeyHash>
    and verifySig(sig, pubKey, txHash))
```

The parameter `txHash` is a hash of some parts of the transaction, and this hash is the message that was signed to generate the signature. This ensures that these parameters of the transaction can not change without a new signature. The script in Bitcoin looks quite a bit different though, because Bitcoin uses its own custom scripting language.

7.3 Scripting

The scripting language used in Bitcoin is a stack-based language similar to Forth. It is intentionally not Turing-complete so that it can be executed without concern for whether the script will hang. It works like a reverse polish notation calculator. The script is read from left to right. When a value is encountered in the script, it is pushed onto a stack. When an operator is encountered in the script, some values are popped off the stack, the operator is applied to these values, and the result is pushed onto the stack.

There are only a few operators that are used in standard transactions.

- OP_DUP - Pop one value off the stack and push two copies of it back onto the stack.
- OP_HASH160 - Pop one value off the stack, apply a hash function, and push the hash onto the stack.

- OP_EQUALVERIFY - Pop two values off the stack, if they are equal, do nothing, if they are not equal, abort the script with return code false.
- OP_CHECKSIG - Pop two values off the stack and assume that the first is a public key and the second is a signature. Check the signature using the public key, assuming the message signed was a simplified transaction with some parts removed. This operator has direct access to the transaction as if it were a hidden global variable in the script. If the signature is verified, push `True` onto the stack, otherwise push `False` onto the stack.

A standard transaction uses the following scripts.

```
scriptPubKey = "OP_DUP
    OP_HASH160 <pubKeyHash>
    OP_EQUALVERIFY OP_CHECKSIG"

scriptSig = "<sig> <pubKey>"
```

where ⟨sig⟩ is the signature of a simplified transaction, ⟨pubKey⟩ is the owner's public key, and ⟨pubKeyHash⟩ is the address that the bitcoins were previously sent to (and currently belong to). For a valid transaction, ⟨pubKeyHash⟩ should be the same as the hash of ⟨pubKey⟩ to guarantee that the person spending the bitcoins is their true owner. Concatenating the scripts, we get

```
script = "<sig> <pubKey> OP_DUP
    OP_HASH160 <pubKeyHash>
    OP_EQUALVERIFY OP_CHECKSIG"
```

The table below shows the contents of the stack as this script is being evaluated.

Next Token	Stack
⟨sig⟩	(Empty)
⟨pubKey⟩	⟨sig⟩
OPxDUP	⟨sig⟩ ⟨pubKey⟩
OPxHASH160	⟨sig⟩ ⟨pubKey⟩ ⟨pubKey⟩
⟨pubKeyHash⟩	⟨sig⟩ ⟨pubKey⟩ ⟨hash(pubKey)⟩
OPxEQUALVERIFY	⟨sig⟩ ⟨pubKey⟩ ⟨hash(pubKey)⟩ ⟨pubKeyHash⟩
OPxCHECKSIG	⟨sig⟩ ⟨pubKey⟩
(None)	True

Since the script evaluates to True, the transaction passes this stage of the validation.

Chapter 8
The Block Chain

8.1 The Byzantine Generals' Problem

After verification, transactions are relayed to other nodes in the peer-to-peer network. The other nodes will repeat the verification and relay the transaction to more nodes. Within seconds, the transaction should reach most of the nodes on the network. Transactions are then held in pools on the nodes awaiting insertion into the block chain, which is a public record of all transactions that have ever occurred in the Bitcoin network. The block chain is not just a simple list of transaction receipts though. It is specially designed to solve the double-spending problem for a peer-to-peer network of untrusted nodes. As discussed in the Double-Spending section, the goal is to determine the chronological ordering of transactions so that the first payment can be accepted and the second payment can be rejected. So the peer-to-peer network has to have a method to agree on an ordering of transactions even though some peers might be trying to sabotage the system.

This problem of coordinating a group of peers with possible saboteurs is known as the Byzantine Generals' Problem. The name comes from a 1980 paper[20] that described a situation in which the generals of the Byzantine Empire's Army had to agree on whether to launch their attack on the enemy. The generals could communicate by messenger, but any of the generals could be a traitor trying to sabotage the attack. If a treasonous general or group of generals could confuse enough honest generals about the outcome of the decision, the army would be fragmented and meet with defeat. The challenge is to develop a protocol that ensures that the majority's decision is heard by all generals even when not every general can communicate directly with every other general.

This is just like how all the nodes in the Bitcoin network have to know the majority's decision about the chronological ordering of transactions, even though not all nodes can communicate directly and some nodes might be malicious attackers.

8.2 The Solution

The block chain solves the Byzantine Generals' Problem by using computational power as a voting system. First, nodes group transactions into *blocks* that are linked to form the block chain. Nodes broadcast blocks to the entire peer-to-peer network upon creation. Each block contains a hash of the previous block in the chain. Therefore, at the time a block was created, the previous block must have already existed or else the hash would be invalid and the block would be rejected. This provides verifiable proof of the chronological ordering of the blocks in the chain. But if blocks could easily be added by anyone, an attacker could just make an alternate chain that reassigns ownership arbitrarily.

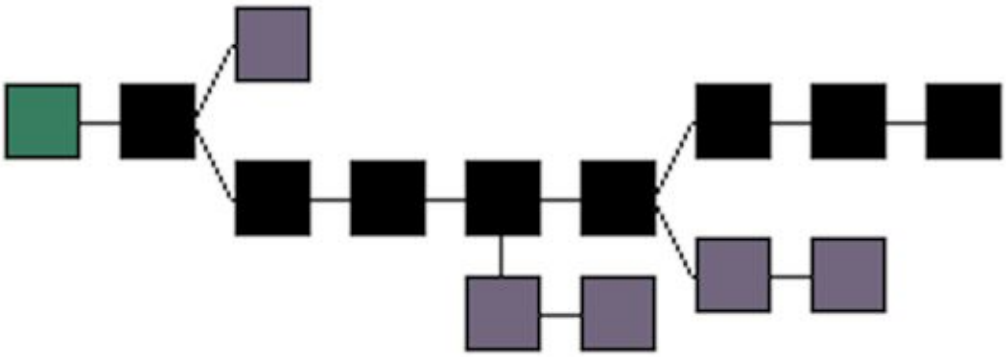


Figure 8.1: The black blocks represent the accepted branch, which is the longest. Gray blocks are kept in the block chain, but ignored because they are not on the longest branch. The leftmost block is the first block in the block chain, which is called the *genesis block*.

To prevent this, Bitcoin nodes require that a difficult proof of work problem be solved in order to add a block to the block chain. Since each block depends on a previous block, this creates a proof of work chain. To replace the entire chain, one would have to solve the proof of work problem for every block in the chain, which would require a massive amount of computing power. An attacker could still try to rewrite the transaction log by creating an alternate block to substitute the most recent block. In this case, there would be two blocks that both point to the same previous block, creating a fork in the block chain. This can also happen if two honest nodes coincidentally create a block at about the same time. If both blocks were kept, it could create ambiguity about the ordering of the transactions, so one has to be chosen. Bitcoin nodes always

choose the "longest" chain (see figure)[13], where length is defined as total computational difficulty. If there is a fork at the top of the chain, then both branches of the fork would have the same length. In this case, each node chooses the block that it received first and works on trying to extend the chain from that block. When the next block is created, it will point to one branch of the fork and resolve the tie by making that branch the longest. When a block fails to make it into the longest chain, the transactions in the block are recycled into the pool awaiting processing, so they are not lost.

The branch that wins is most likely to be the branch that the most nodes are working on. If an attacker chooses to work on an alternate branch, the attacker would have to have as much computing power as the rest of the network in order to keep up with the growth of the main branch. If the alternate branch's growth doesn't keep up, it wouldn't be the longest and it would get ignored by all the honest nodes. Effectively, computing power acts as a vote for determining which blocks to choose. This is how Bitcoin solves the Byzantine Generals' Problem.

8.3 Criticisms

The block chain is not really an easily scalable solution. All nodes have to see every transaction in the world and the block chain is stored in full on every node. In other words, Bitcoin is not a distributed system, it's just massively replicated. Future versions of the Bitcoin software may make it possible to reduce the storage requirements for the block chain, but collecting all unprocessed transactions on every node is not a requirement that can be easily removed. It works fine currently, but if Bitcoin becomes more mainstream, scalability could be an issue. It may require such an expensive computer to operate a node that individuals wouldn't be able to do it. At that point, Bitcoin loses some of its major benefits. For example, it would be easier for governments to regulate. Consider a situation in which a small number of companies control the majority of computational power in the Bitcoin network. The government could easily regulate these companies and force them to install software updates that give the government backdoors to manipulate the currency.

Chapter 9
Mining

9.1 Procedure

The process of creating new blocks is called *mining*, and nodes that do mining are called *miners*. Mining consists of the following steps, performed in a continuous loop:

1. Collecting transactions that were broadcast on the peer-to-peer network into a block. Each miner can arbitrarily decide which transactions to include in their block. Transactions typically have a fee that the miner will receive if their block is accepted, so miners have an incentive to include as many transactions as possible, up to the 1 megabyte block size limit.[14]
2. Verifying that all transactions in the block are valid (as explained in section 7.2).
3. Selecting the most recent block on the longest path in the block chain and inserting a hash of its header into the new block (as explained in section 8.2).
4. Trying to solve the proof of work problem (discussed in the next section) for the new block and simultaneously watching for new blocks coming from other nodes.
    - If a solution is found to the proof of work problem, the new block is added to the local block chain and broadcasted to the peer-to-peer network.
    - If another node solves the proof of work problem first (most likely), the proof of work and the transactions in their block are checked for validity. If the checks pass, their block is added to the local copy of the block chain and relayed on the network; otherwise their block is discarded.

All miners are trying to create new blocks at the same time, with almost all their time being spent on the proof of work problem. Since nodes cannot communicate instantaneously, different nodes may have slightly different versions of the block chain at any given instant, but under normal circumstances the vast majority of nodes will be in agreement. This is because new blocks are created at a regulated rate of one every 10 minutes or so (see

section 9.3, "Difficulty Targeting"), whereas propagation of a new block only takes seconds. So normally there is a period of (on average) 10 minutes where all the nodes are churning through the proof of work problem, until one random node is lucky enough to solve it. Then the new block is broadcast and within seconds all the nodes accept it and start working on the following block. When two nodes coincidentally create a new block at about the same time, the nodes may be split for a while on which one to use, but the "longest chain rule" will ensure that a consensus is re-established quickly (see the Block Chain chapter).

There is no guarantee that all miners will be processing the same set of transactions at any given time because miners select transactions arbitrarily for inclusion into their blocks. However, it is likely that there will be a lot of overlap in the sets of transactions that miners are processing since each miner will try to include as many as possible. So when a new block is added to the block chain, some portion of the transactions that were being processed may not have made it into the new block. These transactions are kept in the pool of unprocessed transactions so that miners can choose to include them in the next block.

9.2 Proof of Work

Miners search for acceptable blocks using the following procedure, performed in a loop:

1. Increment (add 1 to) an arbitrary number in the block header called a *nonce*.
2. Take the hash of the resulting block header (see CBlockHeader, shown below).
3. Check if the hash of the block header, when expressed as a number, is less than a predetermined target value.

If the hash of the block header is not less than the target value, the block will be rejected by the network. Finding a block that has a sufficiently small hash value is the proof of work problem, just as in Adam Back's Hashcash (see section 5.3). The target for Hashcash was a certain number of zeroes at the beginning of the binary representation of the hash, which means that all targets are powers of two. Bitcoin generalizes this by allowing targets to be numbers that start with zeroes followed by a specified pattern of binary digits, not all of which have to be zero.

Because of the properties of cryptographic hash functions, there is no better way to find a solution than this brute-force method. The proof of work is a useless computation aside from the fact that it gives a probabilistically predictable cost to block creation.

```
class CBlockHeader
    int nVersion;
    uint256 hashPrevBlock;
    uint256 hashMerkleRoot;
    unsigned int nTime;
    unsigned int nBits;
    unsigned int nNonce;

class CBlock : public CBlockHeader
    std::vector<CTransaction> vtx;
```

`nVersion` is a version number to support multiple concurrent versions on the network. `hashPrevBlock` is a double SHA-256 hash of the header of the previous block in the block chain. This is the "pointer" that links blocks and defines the chain structure. `hashMerkleRoot` is the top hash of the Merkle tree for all the transactions in the block (all transactions in `vtx`). `nTime` is the approximate time when the block was created, specified as a unix timestamp (seconds since the first second of the year 1970). `nBits` is a compressed representation of the target for the proof of work. `nNonce` is the nonce that is incremented when trying to solve the proof of work problem.

9.3 Difficulty Targeting

It is important that new blocks are not created too quickly or too slowly. If blocks are continually created faster than the time it takes for them to be distributed over the network, then the block chain will become full of forks. Too many forks make it much harder for nodes to reach a consensus on which branch of the block chain to use. On the other hand, if blocks are created too slowly, it takes too long for transactions to be confirmed.

Over time, as more miners start mining and as hardware speed increases, the rate of solving the proof of work problem for a given difficulty level will likely increase. This growth can’t be accurately predicted, so Bitcoin nodes actively regulate the rate of creation of new blocks so that it takes an average of 10 minutes to create each block. The regulation is done by periodically adjusting the hash target value for blocks. Since the hash of the block’s header

must be less than the target, a smaller target makes it harder to find acceptable blocks.

Every 2016 blocks (which spans 2 weeks if each block takes 10 minutes), the nodes calculate a new difficulty based on the time it took to mine the last 2016 blocks. A hash value is a number with 256 bits, so there are 2^{256} possible hash values. The probability of finding a hash value less than T in a single trial is

$$p(T) = T/2^{256}$$

The expected number of trials needed to find a hash less than a target T is

$$N(T) = 1/p(T) = 2^{256}/T$$

If the last 2016 blocks were mined in an average time of t_{avg} using a target of T, then we can estimate that the miners were collectively hashing at a rate of about

$$r_{est} = N(T)/t_{avg}$$

The expected time to discover a block for a difficulty T and hash rate r is

$$t(T, r) = N(T)/r$$

So the target should be adjusted to T' so that $t(T', r_{est}) = 600s$, or 10 minutes.

$$600s = t(T', r_{est}) = N(T')/r_{est}$$

$$= N(T')/(N(T)/t_{avg}) = t_{avg} * (T/T')$$

$$T' = \frac{t_{avg}}{600s} T$$

This target is not always used though because there is an additional rule that the target can only change by a factor of 4 up or down in an single retargeting.

The *difficulty D* is defined as the maximum target over the current target, where the maximum target is $65535 * 2^{208}$,

$$D = T_{max}/T = 65535 * 2^{208}/T$$

9.4 Reward

Solving the proof of work problem requires a lot of computing power and that power costs money. To encourage people to invest their resources in mining, Bitcoin provides a reward in each successfully mined block. As the first transaction in each block, miners insert a *coinbase transaction* (also known as a *generation transaction*) that pays a reward to themselves. The coinbase transaction has exactly one input, but the `prevout` field of the input is set to `NULL` because it is creating new bitcoins, not transferring them. The output of the coinbase transaction specifies one of the miner's addresses so that they can receive their reward if their block makes it into the accepted branch of the block chain. Since the accepted branch of the block chain can change, it isn't possible to know immediately whether a new block will stay on the accepted branch. This is why Bitcoin requires a 100 block *maturation time* before coinbase transactions can be spent. So unlike normal transactions that take about an hour to process, coinbase transactions take about 17 hours.

The reward system is also the exclusive way that new bitcoins are released into circulation. Using this mechanism to release new bitcoins provides a slow and steady rate of production and distributes them in proportion to investment in the system. If all bitcoins were created in one batch at the beginning, then Satoshi Nakamoto would have owned them all and he would probably have had a hard time selling them.

Initially, the reward was 50 bitcoins per block. But if that continued forever, the currency would perpetually experience inflation due to an increasing money supply. Therefore, Bitcoin halves the size of the reward every 210,000 blocks, which is approximately every 4 years since each block takes an average of 10 minutes to mine (4 years * 365 days/year * 24 hours/day * 6 blocks/hour = 210240).[21] Therefore there are 210,000 payments at each reward level, making a total of

$$210,000 * (50 + 25 + 12.5 + 6.25 + 3.125 + ...)$$
$$= 210,000 * 100 = 21,000,000$$

After 21 million bitcoins have been produced, production will stop and remain at this level forever. At this point miners will still receive *transaction fees* on each transaction in blocks that they mine. Transaction fees are an optional way for a sender to increase the speed at which their transaction will be processed by providing an incentive for miners to include the transaction in their next block. When creating a transaction, any excess of the value of the inputs over the value of the outputs is considered a transaction fee that goes to whichever miner processes the transaction. Currently, the total value of transaction fees is much smaller than the value of the coinbase transaction, but as the coinbase transaction's value diminishes, transaction fees will provide an increasing portion of mining profits.

9.5 Mining Pools

Mining works like a lottery where a unit of computing power corresponds to a lottery ticket. As in the lottery, with only one ticket (one unit of of computing power, say a desktop PC doing 10 million hashes per second) it requires a lot of luck to get any reward at all. But the more tickets/computing power one has, the better the odds become. To earn more steady rewards, miners sometimes form pools that collaborate on solving the proof of work problem and share the rewards.

Mining pools are coordinated by a central server that assigns work to miners and distributes rewards to all pool members when any miner in the pool solves the proof of work for a new block. The main challenge of a pool server is to fairly calculate the percentage of the reward to give to each member. In a fair pool, members that contribute more computing power should get paid more, so contributed computing power is measured and tracked by the server. This tracking is accomplished by recording the number of solutions miners find to an easier version of the proof of work problem: the same problem, but with a higher target (lower difficulty). This simpler problem proves that the miner was working on the proof of work problem. Each solution submitted to the server for this easier proof of work problem earns the miner a *share*. The more computing power a miner contributes, the more frequently they will find solutions and earn shares. When a miner in the pool solves the real proof of work problem, the server distributes the reward to all miners in proportion to how many shares they earned since the last reward payout (sometimes with a weighting factor, see below).

There are many different types of pool servers, but a simple one might work like this:

- The pool server prepares a block with the coinbase transaction pointing to the pool's address.
- Miners in the pool contact the pool server and make a `getwork` request to get the block to work on.
- Each miner tries to solve the proof of work problem for the block by incrementing the nonce and hashing the block header.
- Whenever a miner finds a hash value that is below the easier target, it submits the solution to the server for a share.
- The mining server verifies submitted shares and tracks how many each miner has.
- When a miner finds a solution to the proof of work problem, the server pays out the reward in proportion to the number of shares each miner earned since the last payout.
- Miners periodically contact the pool server for updates on what to work on in case a new block was discovered.

If a miner tries to cheat by submitting shares while working on blocks that pay to their own address, the pool server will detect this and reject the shares. The server only accepts block header solutions that correspond to the ones it issues. The only fields that miners can change are `nNonce` and on some servers `nTime`. If the miner changes the destination of the coinbase transaction, the `hashMerkleRoot` field will change and the server will know that the miner is cheating.

To prevent duplicate work, a pool server should have a unique response to every `getwork` request. Otherwise multiple miners will be checking the same nonce values because each miner is just going to increment the nonce starting from zero. To ensure uniqueness, the server can tweak the `nTime` field by a few seconds or reorder the transactions, which would change the `hashMerkleRoot`.

There are several ways to optimize the pool mining system above. One inefficiency arises from the fact that miners only request new work from the server periodically, and a new block could be discovered within that period. Any work done for a block that was already discovered is a waste. A method known as *long polling* optimizes this by having the server contact the miners when a new block is found. Long polling can also reduce network traffic

because miners can continue working until the entire nonce search space is covered without contacting the server.

Pools that pay in direct proportion to the number of shares submitted, as described above, have different profitability for miners at different times. The reason is that if it takes a longer time to find a solution, more shares will be submitted, and the payout per share will be lower. So it is more profitable for a miner to spend more time mining in short *rounds*, where a round is the interval between payouts in a given pool. This encourages *pool hopping*, where miners hop from one pool to another to try to boost their profits. The optimal strategy to exploit proportional-paying pools is to switch to another pool when the number of shares in the round exceeds 43.5% of the current difficulty, assuming each share has difficulty of 1.[24] If a miner follows this strategy, they will increase the percentage of their time that they are working in shorter rounds because they actively leave rounds when they start to become long. Since shorter rounds pay more per share, this maximizes the payout per share. Many pools now have adjustments that discourage pool hopping by making later shares worth more.

9.6 Mining Hardware

Initially, Satoshi's Bitcoin client did mining on a user's PC, but now CPUs have been eclipsed by more efficient mining hardware. GPUs (Graphics Processing Unit - Graphics cards) are designed for doing lots of simple calculations in parallel and are orders of magnitude faster than CPUs. Recently, ASICs (Application-Specific Integrated Circuits) have been developed that are orders of magnitude faster than GPUs. At this point, miners need to have custom hardware to make mining a profitable investment.

Bibliography

[1] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," referenced in e-mail sent to cryptography@metzdowd.com mailing list, October 31, 2008. http://www.mail-archive.com/cryptography@metzdowd.com/msg09959.html

[2] "Satoshi Nakamoto," Bitcoin Wiki, June 13, 2013. https://en.bitcoin.it/wiki/Satoshi_Nakamoto#Work

[3] "Bitcoin Ladder," Bitcoin Wiki, June 14, 2013. https://en.bitcoin.it/wiki/Bitcoin_Ladder#Top_companies

[4] Andy Greenberg, "Black Market Drug Site 'Silk Road' Booming: $22 Million In Annual Sales," Forbes, August 6, 2012. http://www.forbes.com/sites/andygreenberg/2012/08/06/black-market-drug-site-silk-road-booming-22-million-in-annual-mostly-illegal-sales/

[5] Reuben Grinberg, "Bitcoin: An Innovative Alternative Digital Currency," Hastings Science & Technology Law Journal (4) (2011): 160. http://ssrn.com/abstract=1817857

[6] Ralph Merkle, "A certified digital signature," Proceedings on Advances in cryptology (CRYPTO '89) (1989): 218-238. Originally submitted to CACM, 1979. http://www.merkle.com/papers/Certified1979.pdf

[7] Claude Shannon, "Communication Theory of Secrecy Systems," Bell System Technical Journal 28 (4) (1949): 656715.

[8] Satoshi Nakamoto, e-mail to cryptography@metzdowd.com mailing list, November 14, 2008. http://www.mail-archive.com/cryptography@metzdowd.com/msg10001.html

[9] Adrianne Jeffries, "Four years and $100 million later, Bitcoin's mysterious creator remains anonymous," The Verge, May 6, 2013. http://www.theverge.com/2013/5/6/4295028/report-satoshi-nakamoto

[10] Timothy Lee, "An Illustrated History Of Bitcoin Crashes," Forbes, April 11, 2013. http://www.forbes.com/sites/timothylee/2013/04/11/an-illustrated-history-of-bitcoin-crashes/

[11] Laurie Law, Susan Sabett, Jerry Solinas, "How to make a mint: the cryptography of anonymous electronic cash," National Security Agency, Office of Information Security Research and Technology, Cryptology Division, June 18, 1996. http://groups.csail.mit.edu/mac/classes/6.805/articles/money/nsamint/nsamint.htm

[12] David Chaum, "Blind signatures for untraceable payments," Advances in Cryptology Proceedings of Crypto 82 (3) (1983): 199-203.

[13] David Chaum, Amos Fiat, Moni Naor, "Untraceable Electronic Cash," CRYPTO '88 Proceedings on Advances in cryptology (1990): 319-327. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.5759

[14] Ian Grigg, "Triple Entry Accounting," December 25, 2005. http://iang.org/papers/triple_entry.html

[15] Wei Dai, "b-money," referenced in e-mail to cypherpunks mailing list, November 27, 1998. http://www.weidai.com/bmoney.txt

[16] Adam Back, "Hashcash - a denial of service counter-measure," August 1, 2002. http://www.hashcash.org/papers/hashcash.pdf

[17] Cynthia Dwork, "Pricing via Processing, Or, Combating Junk Mail, Advances in Cryptology," CRYPTO '92 Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology (1993): 139-147. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.83.7634

[18] Nick Szabos, "Bit gold," December 27, 2008. http://unenumerated.blogspot.com/2005/12/bit-gold.html

[19] "Protocol specification," Bitcoin Wiki, May 5, 2013. https://en.bitcoin.it/wiki/Protocol_specification#Addresses

[20] Leslie Lamport, Robert Shostak, Marshall Pease, "The Byzantine Generals Problem," ACM Transactions on Programming Languages and Systems 4 (3) (1982): 382401. http://research.microsoft.com/en-us/um/people/lamport/pubs/byz.pdf

[21] "Why was 21 million picked as the number of bitcoins to be created?" Bitcoin Stack Exchange, March 7, 2013. http://bitcoin.stackexchange.com/questions/8439/why-was-21-million-picked-as-the-number-of-bitcoins-to-be-created

[22] "Technical background of Bitcoin addresses," Bitcoin Wiki, March 14, 2013. https://en.bitcoin.it/wiki/Technical_background_of_Bitcoin_addresses

[23] "Why are Bitcoin addresses hashes of public keys?" Bitcoin Stack Exchange, May 8, 2012. http://bitcoin.stackexchange.com/questions/3600/why-are-bitcoin-addresses-hashes-of-public-keys

[24] Raulo, "Optimal pool abuse strategy," February 4, 2011. http://bitcoin.atspace.com/poolcheating.pdf

Notes

1Monetary inflation is a sustained increase in the supply of money, which typically results in price inflation. It is a serious risk factor for fiat currencies because governments often produce money excessively, causing perpetual price inflation.

2The creator of Bitcoin defines a bitcoin as a "chain of digital signatures" in the public ledger known as the block chain.[1]

3The Bitcoin source code can be found at https://github.com/bitcoin/bitcoin

4According to the Bitcoin Wiki, the second biggest bitcoin based company is the underground drug website known as the Silk Road.[3] The sales figures were estimated by Carnegie Mellon computer security professor Nicolas Christin.[4] Six of the other businesses in the top 20 largest are gambling related.[3]

5The chart is from bitcoincharts.com

6See https://en.bitcoin.it/wiki/Tor

7These are foreign exchange fees, not Bitcoin transaction fees (which are much smaller).

8There is a 1 in 4.29 billion chance that a mistyped address passes the checksum test. If this happens, your money could be sent to the wrong person, or much more likely be sent to an address that nobody owns, which effectively eliminates the bitcoins from circulation.

9There are also several more subjective properties that are important for the success of the system: portability, scalability, acceptability, and reliability. Additionally, offline systems can optionally have the properties of Divisibility and Offline Transferability. We will not discuss these further however.

[10]There are other types of addresses and different networks that use different version numbers. For example, there is a pay-to-script feature in Bitcoin that uses version number 5. And the Namecoin network, which is based on a separate block chain, uses version number 52. There are many more, as you can see at https://en.bitcoin.it/wiki/List_of_address_prefixes

[11]The code snippets in this book are from the Bitcoin-Qt source code. They do not show the full contents of each class, just the important parts of the state in each class.

[12]See https://en.bitcoin.it/wiki/Contracts for more information

[13]The figure is from https://en.bitcoin.it/wiki/Block_chain

[14]This limit may be raised in the future if transaction volume increases.